proc manual page - Built-In Commands

NAME

proc — Create a Tcl procedure

SYNOPSIS

proc *name args body*

DESCRIPTION

The **proc** command creates a new Tcl procedure named *name*, replacing any existing command or procedure there may have been by that name. Whenever the new command is invoked, the contents of *body* will be executed by the Tcl interpreter. Normally, *name* is unqualified (does not include the names of any containing namespaces), and the new procedure is created in the current namespace. If *name* includes any namespace qualifiers, the procedure is created in the specified namespace. *Args* specifies the formal arguments to the procedure. It consists of a list, possibly empty, each of whose elements specifies one argument. Each argument specifier is also a list with either one or two fields. If there is only a single field in the specifier then it is the name of the argument; if there are two fields, then the first is the argument name and the second is its default value. Arguments with default values that are followed by non-defaulted arguments become required arguments; enough actual arguments must be supplied to allow all arguments up to and including the last required formal argument.
When *name* is invoked a local variable will be created for each of the formal arguments to the procedure; its value will be the value of corresponding argument in the invoking command or the argument's default value. Actual arguments are assigned to formal arguments strictly in order. Arguments with default values need not be specified in a procedure invocation. However, there must be enough actual arguments for all the formal arguments that do not have defaults, and there must not be any extra actual arguments. Arguments with default values that are followed by non-defaulted arguments become de-facto required arguments, though this may change in a future version of Tcl; portable code should ensure that all optional arguments come after all required arguments.

There is one special case to permit procedures with variable numbers of arguments. If the last formal argument has the name "**args**", then a call to the procedure may contain more actual arguments than the procedure has formal arguments. In this case, all of the actual arguments starting at the one that would be assigned to **args** are combined into a list (as if the **list** command had been used); this combined value is assigned to the local variable **args**.

When *body* is being executed, variable names normally refer to local variables, which are created automatically when referenced and deleted when the procedure returns. One local variable is automatically created for each of the procedure's arguments. Other variables can only be accessed by invoking one of the **global**, **variable**, **upvar** or **namespace upvar** commands. The current namespace when *body* is executed will be the namespace that the procedure's name exists in, which will be the namespace that it was created in unless it has been changed with **rename**.

The **proc** command returns an empty string. When a procedure is invoked, the procedure's return value is the value specified in a **return** command. If the procedure does not execute an explicit **return**, then its return value is the value of the last command executed in the procedure's body. If an error occurs while executing the procedure body, then the procedure-as-a-whole will return that same error.

EXAMPLES

This is a procedure that takes two arguments and prints both their sum and their product. It also returns the string “OK” to the caller as an explicit result.

```
proc printSumProduct {x y} {
    set sum [expr {$x + $y}]
    set prod [expr {$x * $y}]
    puts "sum is $sum, product is $prod"
    return "OK"
}
```

This is a procedure that accepts arbitrarily many arguments and prints them out, one by one.

```
proc printArguments args {
    foreach arg $args {
        puts $arg
    }
}
```

This procedure is a bit like the **incr** command, except it multiplies the contents of the named variable by the value, which defaults to **2**:

```
proc mult {varName {multiplier 2}} {
    upvar 1 $varName var
    set var [expr {$var * $multiplier}]
}
```